

Variations on Memetic Algorithms for Graph Coloring Problems

Laurent Moalic^{*1} and Alexandre Gondran^{†2}

¹IRTES-SET, UTBM, University of Technology of Belfort-Montbéliard, Belfort, France

²MAIAA, ENAC, French Civil Aviation University, Toulouse, France

Abstract

Graph vertices coloring with a given number of colors is a famous and much-studied NP-complete problem. The best methods to solve this problem are hybrid algorithms such as memetic algorithms [15, 26, 38] or quantum annealing [34, 35, 36]. Those hybrid algorithms use a powerful local search inside a population-based algorithm. The balance between intensification and diversification is essential for those metaheuristics but difficult to achieve. Customizing metaheuristics takes long time and is one of the main weak points of these approaches. This paper studies the impact of the increase and the decrease of diversification in one of the most effective algorithms known: the Hybrid Evolutionary Algorithm (HEA) from Galinier and Hao [15]. We then propose a modification of this memetic algorithm in order to work with a population of only two individuals. This new algorithm more effectively manages the correct ‘dose’ of diversification to add into the included local search - TabuCol [20] in the case of the HEA. It has produced several good results for the well-known DIMACS benchmark graphs, such as 47-colorings for DSJC500.5, 82-colorings for DSJC1000.5, 222-colorings for DSJC1000.9 and 81-colorings for flat1000_76_0, which have so far only been produced by quantum annealing [36] in 2012 with massive multi-CPUs.

1 Introduction

Given a undirected graph $G = (V, E)$ with V a set vertices and E a set of edges, the graph vertex coloring involves assigning each vertex with a color so that two adjacent vertices (linked by an edge) feature different colors. The Graph Vertex Coloring Problem (GVCP) involves finding the minimum number of colors required to color a given graph in respect of these binary constraints. The GVCP is a famous and much-studied problem because this simple formalization can be applied to various issues such as

^{*}laurent.moalic@utbm.fr

[†]alexandre.gondran@enac.fr

frequency assignment problems [1, 10], scheduling problems [25, 39, 37] and fly level allocation problems [3]. Most problems that involve sharing a rare resource (colors) between different operators (vertices) can be modeled as a GVCP, such as most resource allocation problems. GVCP is NP-hard [18]. Given k a positive integer corresponding to the maximum number of colors, a k -coloring of a given graph G is a function c that assigns to each vertex a color (i.e. an integer included between 1 and k) as follows :

$$\begin{aligned} c : V &\rightarrow \{1, 2, \dots, k\} \\ v &\mapsto c(v) \end{aligned}$$

One recalls some definitions : a k -coloring is called *legal* or *proper k -coloring* if it respects the following binary constraints : $\forall (u, v) \in E, c(u) \neq c(v)$. Otherwise the k -coloring is called *non legal* or *non proper*; and edges $(u, v) \in E$ such as $c(u) = c(v)$ are called *conflicting edges*, and u and v *conflicting vertices*. A given graph G is *k -colorable* if a proper k -coloring exists. The *chromatic number* $\chi(G)$ of a given graph G is the smallest integer k such as G is k -colorable. A coloring is called *complete coloring* if one color is assigned to *all* vertices, otherwise it is called *partial coloring*, whereby some vertices may remain uncolored. An *independent set* or *stable set* is a set of vertices, no two of which are adjacent. In this case, it is possible to assign the same color to all the vertices of an independent set without producing any conflicting edges. The problem of finding a minimal graph partition of independent sets is then equivalent to the GVCP.

The k -coloring problem - finding a proper k -coloring of a given graph G - is NP-complete [24] for $k > 2$. Therefore, the best performing exact algorithms are generally not able to find a proper k -coloring in reasonable time when the number of vertices is greater than 100 [22, 11]. For large graphs, one uses heuristics that partially explore the search space to occasionally find a proper k -coloring in a reasonable time frame. However, this partial search does not guarantee that a better solution exists. Very interesting and comprehensive surveys on the GVCP and the most effective heuristics to solve it can be found in [16, 14]. These studies classify heuristics by the search space used. In order to define the search space (or that which is termed strategy), one has to answer to three questions :

1. Is the number of available colors fixed or not ?
2. Is non proper colorings included in the search space ?
3. Does the heuristic use complete colorings or partial colorings ?

Among the eight theoretical possibilities, four main strategies are defined [14] :

- *Proper strategy* if the number of colors is not fixed and only complete and proper colorings are taken into account. The aim is to find a coloring that minimizes the number of colors used under constraints of legality and completeness of the coloring.
- *k -fixed partial proper strategy* if the number of colors is fixed and partial and proper colorings are taken into account. The aim is to find a coloring that minimizes the number of uncolored vertices under constraints of the number of given colors and of proper coloring.

- *k-fixed penalty strategy* if the number of colors is fixed and complete and no proper colorings are taken into account. The aim is to find a coloring that minimizes the number of conflicting edges under constraints of the number of given colors and of completed coloring.
- *Penalty strategy* if the number of colors is not fixed and no proper and completed colorings are taken into account. The aim is to find a coloring that minimizes the number of conflicting edges and the number of colors used under the constraint of completed coloring.

The Variable Space Search of [21] is interesting and didactic because it works with three of the four above strategies. Another more classical means of classifying the different methods is to consider how these methods explore the search space.

- Constructive or exhaustive methods (such as greedy methods, branch and bound, backtracking and constraint programming) build a coloring step-by-step from empty coloring; those approaches usually have a *k-fixed partial proper strategy*. DSATUR [5] and RLF [25] are the most well-known greedy algorithms. Those algorithms rapidly provide an upper bound on $\chi(G)$, but it is quite distance from the exact value of $\chi(G)$. They are used to initialize solutions before a local search or an evolutionary algorithm is employed. Some improved algorithm such as XRFL [22] based on RLF provides much better results. Exact algorithms such as branch and bound (B&B), backtracking, or constraint programming [6] are *k-fixed partial proper strategies*. The B&B implementation of [22] takes too much time after 100 vertices. Column generation approaches are also exact methods. [27] divided the problem into two parts: the first problem involves generating useful columns in the form of independent sets. The second problem involves selecting a minimum number of independent sets (created by the first problem) in order to cover the graph. To obtain better results, exact methods can be hybridized with local searches [32, 9].
- Local (or neighborhood or trajectory) searches (such as the hillclimbing, the simulated annealing [22], the tabu search [20, 8], the variable neighborhood search [2], the variable space search [21] and the min-conflict) start from an initial coloring and try to improve it by local moves; usually those approaches have a *proper* or *k-fixed penalty strategies*. A detailed overview of those local search methods for graph coloring is provided in [16]. In our algorithm, we employ an improved version [15] of a tabu search algorithm called TabuCol [20]; it is one of the first metaheuristics developed for graph coloring and uses a *k-fixed penalty strategy*.
- Population-based approaches (such as the evolutionary algorithm, the ant colony optimization [29], the particle swarm algorithm and the quantum annealing [34, 35]) work with several colorings that can interact together with, for example, a crossover operator, a share memory, a repulsion or attraction operator, or others criteria such as sharing. Those methods currently obtain the best results when they are combined with the aforementioned local search algorithms. However, used alone, those population-based approaches are limited.

The first objective of this paper is to present a new and simple algorithm that provides the best results for coloring DIMACS benchmark graphs. The second objective is to show why this algorithm effectively controls diversification. Indeed, in our work, the population of this memetic algorithm is reduced to only two individuals, called a couple-population. It provides the opportunity to focus on the correct ‘dose’ of diversity to add to a local search. This new approach is simpler than a general evolutionary algorithm because it does not use numerous parameters such as selection, size of population, crossover and mutation rates.

The organization of this paper is as follows. The issue of diversification in heuristics is presented in Section 2. Section 3 describes our improvement of the memetic algorithm for the GVCP. The experimental results are presented in Section 4 and some of the impacts of diversification are analyzed in Section 5. Finally, we consider the conclusions of this study and discuss possible future research in Section 6.

2 How to manage diversity ?

Because the search is not exhaustive, heuristics provide only sub-optimal solutions with no information about the distance between those sub-solutions and the optimal solution. Heuristics return the best solution found after partial exploration of the search space. To produce a good solution, heuristics must alternate exploitation phases and exploration phases. During an exploitation phase (or an intensification phase), a solution produced by the heuristic is improved. It resembles an exhaustive search but occurs in a small part of the search space. An exploration phase (or diversification phase) involves a search into numerous uncharted regions of the search space. The balance between those two phases is difficult to achieve. Metaheuristics therefore define a framework that manages this balance. In this section, we classify the main components of several metaheuristics as intensification operators or as diversification operators. Of course, this list, presented in table 1, is not exhaustive and we take into account only well-known metaheuristics : Local Search (LS), Tabu Search (TS), Simulated Annealing (SA), Variable Neighborhood Search (VNS), Evolutionary Algorithm (EA), and Ant Colony (AC). Some components can be shared between several metaheuristics. Other components can be at the same time an intensification operator and a diversification operator, such as parent selection or population update. It depends on the context of the algorithm, as we will now demonstrate.

Local search algorithms start from an initial solution and then try to improve this iteratively through local transformation. The simplest LS, hill-climbing, accepts a move only if the objective function is improved: it is inherently an intensification operator. It is possible to introduce some diversity by generating several different initial solutions. This process is called multi-starting. The limit of a simple LS is that after a given number of iterations, the algorithm is blocked within a local optimum. No local transformation can improve the solution. SA and TS therefore accept some worsening moves during the search, with a given criteria for SA and with a tabu list for TS. On the other hand, VNS, to escape local optimum, changes the neighborhood structure. The change in neighborhood structure is an effective diversification operator because it

never worsens the current solution. However, this diversification operator is too weak: VNS must add a shaking process, similar to partial restart (as in Iterated Local Search), in order to achieve a more global optimization.

The population-based algorithms are often classified as global optimization algorithms because they work with several candidate solutions, although this does not indicate that the algorithm will find the global optimum. Moreover, working with several candidate solutions can be regarded as a diversification operator. EAs are population-based algorithms. A basic EA can be presented in five steps as follows: 1) Parents selection: one selects two individuals of the population, which are called parents. 2) Crossover: according to a given rate, a crossover operator is applied to those two parents, which creates two new individuals, called children, blending of the two parents. 3) Mutation: according to a given rate, a mutation operator is applied to each children, which modifies them slightly. 4) Population update: under given conditions, the two children take the place of two individuals of the population. 5) This cycle of four steps is called a generation; it is repeated until a given stop condition is realized. The best individual of the population is the output of the EA. We shall classify the different components of this basic EA as intensification operators or diversification operators.

Mutation and crossover operators are different in nature to selection and population update processes. Indeed, mutation and crossover operators are chance generators, while selection and population update processes are higher-level systems that control the chance interest. These two means of functioning are pithily summarized in an expression attributed to Democritus: *“Everything existing in the universe is the fruit of chance and necessity”*, which is also considered in Jacques Monod’s book *Chance and Necessity* [28]. The mutation and the crossover processes are therefore diversification operators in essence. The mutation operator slightly changes one of the individuals of the population. The crossover operator mixes two individuals of the population in order to create a new one. The mutation is a unary operator while the crossover is a binary operator. There exists also trinary operators for some other EAs, such as for Differential Evolution. The unary or binary changes (applied by mutation or crossover) are performed randomly and these are therefore exploration phases. Occasionally, a random modification improves the solution, but this function is driven by the higher-level system. These changes cannot be considered intensification operators, except in some specific cases where the modifications (mutation or crossover) are not performed randomly. For example, in [12], the crossover is a quasi-deterministic process directly guided by the separability of the objective function in order to improve the latter. The aim of a diversification operator is to provide diversity to current solution(s), but there is a risk of providing too much diversity, and thus breaking the structure of current solution(s). The crossover operator generally provides more diversity than the mutation operator.

The classification of operators such as the parents selection and the population update depends on the choice of the EA parameters. Indeed, if one chooses an elitist parents selection policy (the choice of best individuals of the population for the crossover and the mutation), the selection is then an intensification operator. Conversely, a policy of random parents selection (a random selection of individuals from the population for the crossover and the mutation) indicates a diversification operator. This double

Intensification/Exploitation	Diversification/Exploration
LS: accept improving moves	LS: multi-starts
VNS: change of neighborhood structure	SA, TS: accept worsening moves
	VNS: change of neighborhood structure shaking
EA: parents selection (the best)	p-bA: population (several candidate solutions)
population update (if child is better)	EA: parents selection (random)
	population update (systematic)
	mutation, crossover
	sharing, elitism
AC: collective memory (pheromone)	AC: random choice

Table 1: Classification of main components of some well-known metaheuristics as intensification operators or as diversification operators. LS: Local Search, TS: Tabu Search, SA: Simulated Annealing, VNS: Variable Neighborhood Search, EA: Evolutionary Algorithm, AC: Ant Colony, p-bA: population based Algorithm.

role of the parents selection can be very interesting but also very difficult to properly control. The population update has the same feature. If one chooses to include in the population the individuals created by the crossover and/or the mutation (children) only if they are better than the individuals of the population that one removes, then the population update is an intensification operator. Conversely, if one chooses to systematically replace some individuals of the population by the created children (even if they are worse), then the population update is a diversification operator. The sharing and the elitism are two others mechanisms of EAs playing diversification operators roles.

In AC, the interactions between individuals of the population occur through the sharing of a collective memory. The phenomenon mechanism (deposit and evaporation) plays this role, while the random choice of the instantiation of variables plays the role of diversification.

Table 1 summarizes the classification of main components of several metaheuristics as diversification operators or as intensification operators. The separation is not always very clear, such as in the case of EAs, in which parents selection or update population processes play both roles. Controlling the correct balance between intensification and diversification is a difficult step; Much fine tuning is required to obtain good results from heuristics.

An interesting feature of a diversification operator is its ability to explore new areas of the search space without breaking the structure of current solution(s). We recall a quotation of the physiologist Claude Bernard about hormones [4] and that corresponds well with the role of a diversification operator :

“Everything is poisonous, nothing is poisonous, it is all a matter of dose.”

Claude Bernard - 1872

The level or dose of diversity provided by a diversification operator is difficult to evaluate. However, it is possible to compare the diversification operators with their dose of diversity. For example, the mutation operator generally provides less diversity than the crossover operator. For each operator, there are many parameters that affect the dose

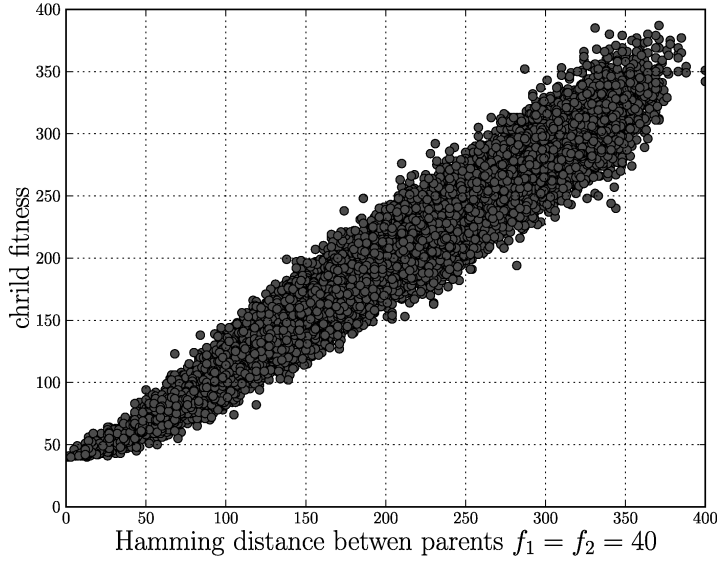


Figure 1: Fitness of the child solution in function of the hamming distance separating the two parents having the same fitness. The fitness is the number of conflicting edges in a completed 48-coloring. The fitness of the parents is equal to $f = 40$. The hamming distance separating two k -colorings is the minimum number of moves (changing of one vertex of color class) to pass from one k -coloring to the other (it is independent of the colors symmetry).

of diversity; One of the easily identifiable parameters for the crossover operator is the Hamming distance between the two parents. Indeed, if we highlight two good solutions that have very good objective functions but that are very different (in terms of Hamming distance), then they would have a high probability of producing children with bad objective functions following crossover. The danger of the crossover is of completely destroying the solution structure. On the other hand, two solutions very closed (in terms of Hamming distance) produce a child with quasi-similar fitness. Chart 1 shows the correlation between the Hamming distance separating the two parents of the same fitness (abscissa axis) and the fitness of the child (ordinate axis). This chart is obtained for k -coloring problem where the objective is to minimize the number of conflicting edges (the fitness) in a complete but non legal k -coloring with $k = 48$. We consider the DSJC500.5 graph from the DIMACS benchmark [23] and the GPX crossover of [15]. The parents used for the crossover have the same fitness value, equal to 40 conflicting edges. There is a quasi-linear correlation between these two parameters.

The key question is : how does one manage the correct ‘dose’ of diversification in a heuristic? In the next section, we present a memetic algorithm with only 2 individuals, which simplifies the management of the diversity dose.

3 Memetic Algorithms with only two individuals

Memetic Algorithms [19] (MA) are hybrid metaheuristics using a local search algorithm inside a population-based algorithm. They can also be viewed as specific EAs where all individuals of the population are local minimum (of a specific neighborhood). This hybrid metaheuristic is a *low-level relay hybrid* (LRH) in the Talbi taxonomy [33]. A low-level hybridization means that a section of one metaheuristic is replaced by a second metaheuristic. In MA, the mutation of the EA is replaced by a local search algorithm. Conversely, a *high-level* hybridization refers to a combination of several metaheuristics, none of which are modified. A *relay* (respectively *teamwork*) hybrid means that metaheuristics are used successively (respectively in parallel).

In graph coloring, the Hybrid Evolutionary Algorithm (HEA) of Galinier and Hao [15] is a MA; the mutation of the EA is replaced by a tabu search. HEA is one of the best algorithms for solving the GVCP; From 1999 until 2012, it provided the majority of the best results for DIMACS benchmark graphs [23], particularly for difficult graphs such as DSJC500.5 and DSJC1000.5 (see table 2). These results were obtained with a population of 8 individuals.

The tabu search used is an improvement of the TabuCol of [20]. This version of TabuCol is a very powerful tabu search, which even obtained very good results for the GVCP when it is used alone (see table 2). Another benefit of this version of TabuCol is that it has only two parameters to adjust in order to control the tabu tenure. Moreover, [15] have demonstrated on a very large number of instances that with the same setting, TabuCol obtained very good k -colorings. Indeed, one of the main disadvantages of heuristics is that the number of parameters to set is high and difficult to adjust. This version of TabuCol is very robust. Thus we retain the setting of [15] in all our tests and consider TabuCol as an atomic algorithm.

The mutation, which is a diversification operator, is replaced by a local search, an intensification operator. The balance between intensification and diversification is restored because in the case of the graph coloring problem, all crossovers are too strong and destroy too much of a solution's structure.

The crossover used in HEA is called the Greedy Partition Crossover (GPX); it is based on color classes. Its aim is to add slightly more diversification to the TabuCol algorithm.

These hybridizations combine the benefits of population-based methods, which are better at diversification by means of a crossover operator, and local search methods, which are better at intensification.

The intensification/diversification balance is difficult to achieve. In order to simplify the numerous parameters involved in EAs, we have chosen to consider a population with only two individuals. This implies 1) no selection process, and 2) a simpler management of the diversification because there is only one diversification operator: the crossover.

3.1 General Pattern - H_2O : Hybrid approach with 2 trajectories-based Optimization

The basic blocks of HEA are the TabuCol algorithm, which is a very powerful local search for intensification, and the Greedy Partion Crossover (GPX), which adds a little more diversification.

The idea is to combine as simply as possible these two blocks. The local search is a unary operator while the crossover is a binary operator. We present with algorithm 1 the pseudo code of a first version of this simple algorithm, which can be seen as a 2 trajectories-based algorithm. We then call the algorithm ' H_2O ', which signifies a Hybrid approach with 2 trajectories-based Optimization.

Algorithm 1 First version of H_2O : Hybrid approach with 2 trajectories-based Optimization

```

1: Input: an asymmetric crossover, a local search.
2: Output: the best solution found best
3:  $p_1, p_2, best \leftarrow \text{init}()$  {initialize with random solutions}
4:  $generation \leftarrow 0$ 
5: repeat
6:    $c'_1 \leftarrow \text{crossover}(p_1, p_2)$ 
7:    $c'_2 \leftarrow \text{crossover}(p_2, p_1)$ 
8:    $c_1 \leftarrow \text{localSearch}(c'_1)$ 
9:    $c_2 \leftarrow \text{localSearch}(c'_2)$ 
10:   $best \leftarrow \text{saveBest}(c_1, c_2, best)$ 
11:   $p_1, p_2 \leftarrow \text{replace}(c_1, c_2)$ 
12:   $generation++$ 
13: until  $\text{stop\_condition}()$ 
14: return best

```

The algorithm 1 needs an asymmetric crossover, which means: $\text{crossover}(p_1, p_2) \neq \text{crossover}(p_2, p_1)$. After randomly initializing the two solutions, the algorithm repeats an instructions loop until a stop criteria occurs. First, we introduce some diversity with the crossover operator, then the two offspring c'_1 and c'_2 are improved by means of the local search. Next, we register the best solution and we systematically replace the parents by the two children. An iteration of this algorithm is called a *generation*.

In order to add more diversity on the algorithm 1, we present a second version of H_2O with two extra elite solutions.

Algorithm 2 Second version of H_2O : Hybrid approach with 2 trajectories-based Optimization

```
1: Input: an asymmetric crossover, a local search,  $Iter_{cycle}$ .
2: Output: the best solution found
3:  $p_1, p_2, elite_1, elite_2, best \leftarrow \text{init}()$  {initialize with random solutions}
4:  $generation \leftarrow 0$ 
5: repeat
6:    $c'_1 \leftarrow \text{crossover}(p_1, p_2)$ 
7:    $c'_2 \leftarrow \text{crossover}(p_2, p_1)$ 
8:    $c_1 \leftarrow \text{localSearch}(c'_1)$ 
9:    $c_2 \leftarrow \text{localSearch}(c'_2)$ 
10:   $elite_1 \leftarrow \text{saveBest}(c_1, c_2, elite_1)$ 
11:   $p_1, p_2 \leftarrow \text{replace}(c_1, c_2)$ 
12:   $best \leftarrow \text{saveBest}(elite_1, best)$ 
13:  if  $generation \% Iter_{cycle} = 0$  then
14:     $p_1 \leftarrow elite_2$ 
15:     $elite_2 \leftarrow elite_1$ 
16:     $elite_1 \leftarrow \text{init}()$ 
17:  end if
18:   $generation++$ 
19: until  $\text{stop\_condition}()$ 
20: return  $best$ 
```

We add two other candidate solutions (similar to elite solutions), $elite_1$ and $elite_2$, in order to reintroduce some diversity in the couple-population. Indeed, after a given number of generations, the two individuals of the population become increasingly similar within the search space. To maintain a given diversity in the couple-population, the $elite_2$ elite solution replaces one of the population individual after $Iter_{cycle}$ generations i.e. one cycle. $elite_1$ is the best solution found during the current cycle and $elite_2$ the best solution found during the previous cycle. The figure 2 represents the graphic view of the algorithm 2.

3.2 Application to graph coloring : H_2col

H_2col is the application of H_2O to the k -coloring problem with the GPX as crossover and the TabuCol as local search. It uses only one parameter: $Iter_{TC}$, the number of iterations performed by the TabuCol algorithm.

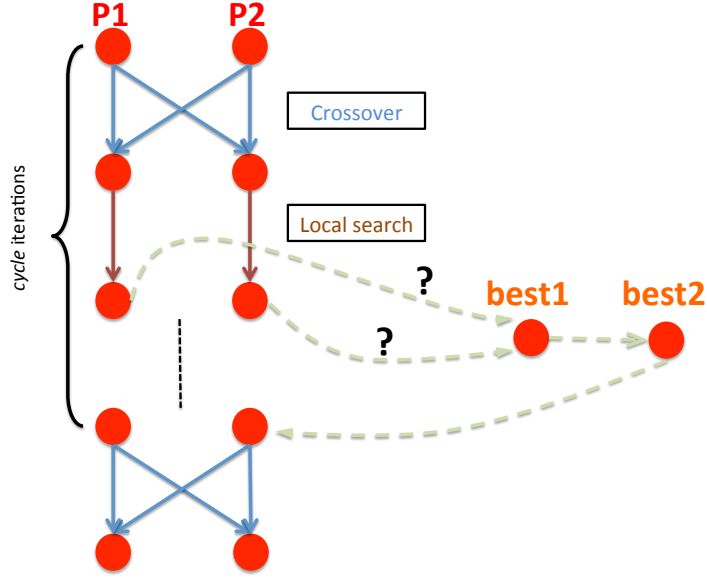


Figure 2: Diagram of H_2O .

Algorithm 3 H_2col : Hybrid approach with 2 trajectories-based Optimization for graph coloring with GPX and TabuCol

- 1: **Input:** the asymmetric crossover: GPX, the local search: TabuCol (with $Iter_{TC}$ iterations), $Iter_{cycle} = 10$.
 - 2: **Output:** the best configuration found
 - 3: $p_1, p_2, elite_1, elite_2 \leftarrow \text{init}()$ {initialize with random colorings}
 - 4: $generation \leftarrow 0$
 - 5: **repeat**
 - 6: $c'_1 \leftarrow \text{GPX}(p_1, p_2)$
 - 7: $c'_2 \leftarrow \text{GPX}(p_2, p_1)$
 - 8: $c_1 \leftarrow \text{TabuCol}(c'_1)$
 - 9: $c_2 \leftarrow \text{TabuCol}(c'_2)$
 - 10: $elite_1 \leftarrow \text{saveBest}(c_1, c_2, elite_1)$
 - 11: $p_1 \leftarrow c_1$
 - 12: $p_2 \leftarrow c_2$
 - 13: $best \leftarrow \text{saveBest}(elite_1, best)$
 - 14: **if** $generation \% Iter_{cycle} = 0$ **then**
 - 15: $p_1 \leftarrow elite_2$
 - 16: $elite_2 \leftarrow elite_1$
 - 17: $elite_1 \leftarrow \text{init}()$
 - 18: **end if**
 - 19: $generation ++$
 - 20: **until** $nbConflicts > 0$
 - 21: **return** $best$
-

This algorithm is an improvement of the TabuCol algorithm; these are two parallel TabuCol algorithms that interact periodically by crossover. We will now briefly recall the principles of the TabuCol algorithm and the GPX crossover.

3.2.1 TabuCol

In 1987, [20] presented the TabuCol algorithm, one year after Fred Glover introduced the tabu search. This algorithm, which solves k -coloring problems, was enhanced in 1999 by [15]. The three basic features of this trajectory-based algorithm are as follows:

- *Search Space and Objective Function:* the algorithm is a k -fixed penalty strategy. The objective function minimizes the number of conflicting edges.
- *Neighborhood:* a k -coloring solution is a neighbor of an other k -coloring solution if the color of only one conflicting vertex is different. This move is called a critic 1-move. Therefore the neighborhood size depends on the number of conflicting vertices.
- *Move Strategy:* the move strategy is the standard tabu search strategy. Even if the objective function is worse, at each iteration, one decides to move to the best neighbors, which are not inside the tabu list. Note that all the neighborhood is explored. If there are several best moves, one chooses one of them at random. This is the only random aspect of this metaheuristic. The tabu list is not the list of each already-visited solution because this is computationally expensive. It is more efficient to put only the reverse moves inside the tabu list. Indeed, the aim is to forbid returning to previous solutions, and it is possible to reach this goal by forbidding the reverse moves during a given number of iterations (i.e. the tabu tenure). The tabu tenure is dynamic: it depends on the neighborhood size. A basic aspiration criteria is also implemented: it accepts a tabu move to a k -coloring, which has a better objective function than the best k -coloring encountered so far.

Data structures have a major impact on algorithm efficiency, constituting one of the main differences between the Hertz and de Werra version of TabuCol [20] and the Galinier and Hao version [15]. Checking that a 1-move is tabu or not and updating the tabu list are operations in constant time (figure 3b). TabuCol also uses an incremental evaluation [13]: the objective function of the neighbors is not computed from scratch, but only the difference between the two solutions is computed. This is a very important feature for local search efficiency. Finding the best 1-move corresponds to finding the maximum value of a matrix (cf. figure 3a).

3.2.2 Greedy Partition Crossover (GPX)

The second block of H_2col is the Greedy Partition Crossover (GPX) from the Hybrid Evolutionary Algorithm HEA [15]. The two main principles of GPX are: 1) a coloring is a partition of vertices and not an assignment of colors to vertices, and 2) large color classes should be transmitted to the child. The figure 4 gives an example of GPX for

	δ -evaluation			
A	+1	X	-2	0
B	-1	+1	X	-1
C	X	0	-2	+1
D	-2	X	-1	+2

	tabu list (iter=21)			
A	15	0	43	20
B	6	34	7	13
C	0	23	10	7
D	32	17	21	0

(a) Matrix of incremental evaluation. It gives the benefit of the all 1-moves; for example, changing the color of the C vertex in green adds 2 conflicts; the cross, for example in (A,blue), means that the A vertex is colored in blue in the current solution; therefore it is not a real 1-move.

(b) Matrix representing the tabu list and the tabu tenure. If the current iteration is the 21st iteration, therefore all the 1-moves of a value greater than 21 means that this 1-move is tabu; for example, (B,blue)-move is tabu because $34 > 21$.

Figure 3: Data structures for TabuCol algorithm.

a problem with three colors (red, blue and green) and 10 vertices (A, B, C, D, E, F, G, H, I and J). The first step is to transmit to the child the largest color class of the first parent. After having withdrawn those vertices in the second parent, one proceeds to step 2 where one transmits to the child the largest color class of the second parent. This process is repeated until all the colors are used. There are most probably still some uncolored vertices in the child solution. The final step is to randomly add those vertices to the color classes.

4 Experimental Results

In this section we present the results obtained with the two versions of the proposed memetic algorithm; the first version of H_2col without elites solutions is indicated as H'_2col , and the second version with the two extra elites solutions is simply indicated H_2col . Test instances are selected among the most studied graphs since the 1990s, which are known to be very difficult (DIMACS [23]). To validate the proposed approach, the results of H_2col are compared with the results obtained by some of the best methods currently known.

4.1 Instances and Benchmarks

We study some graphs from the second DIMACS challenge of 1992-1993 [23]. This is to date the most widely-used benchmark for solving the graph coloring problem. These instances are available at the following address: <ftp://dimacs.rutgers.edu>

We focus on two main types of graphs from the DIMACS benchmark: DSJC and FLAT, which are randomly or quasi-randomly generated. DSJCN.d graphs are graphs with n vertices, with each vertex connected to an average of $n \times d$ vertices; d is the

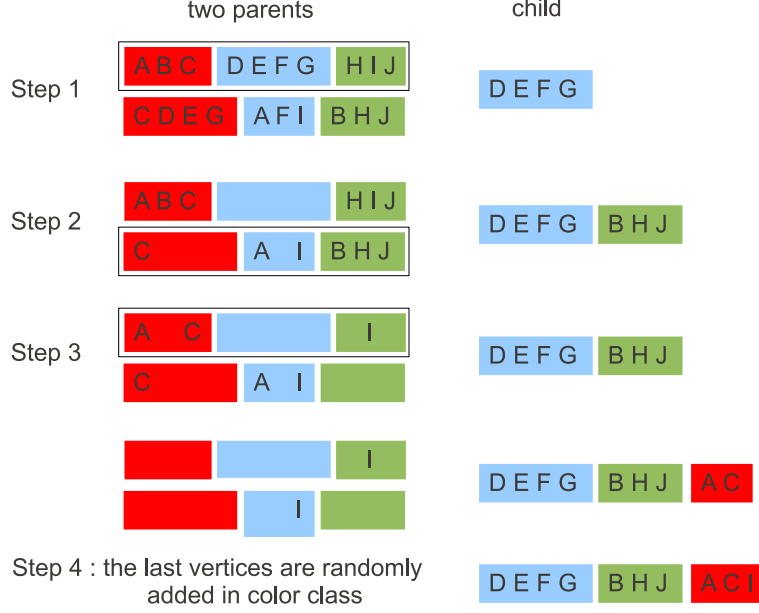


Figure 4: An example of GPX crossover for a graph of 10 vertices (A, B, C, D, E, F, G, H, I and J) and three colors (red, blue and green). This example comes from [15].

graph density. The chromatic number of these graphs is unknown. FLAT graphs have another structure: they are built for a known chromatic number. The $\text{flat}_n\text{-}\chi$ graph has n vertices and χ is the chromatic number.

4.2 Computational Results

$H_2\text{col}$ and $H'_2\text{col}$ were programmed in C++ standard. The results presented in this section were obtained on a computer with an Intel Xeon 3.10GHz processor - 4 cores and 8GB of RAM. Note that the RAM size has no impact on the calculations: even for large graphs such as DSJC1000.9 (with 1000 vertices and high density of 0.9), the memory used does not exceed 125 MB. The main characteristic is the processor speed.

As shown in Section 3, the proposed algorithms have two successive calls to local search (lines 8 and 9 of the algorithms 1, 2 and 3), one for each child of the current generation. Almost all of the time is spent performing the local search. It is possible and moreover easy to parallelize both local searches when using multi-core processor architecture. This is what we have done using the OpenMP API (Open Multi-Processing), which has the advantage of being cross-platform (Linux, Windows, MacOS, etc.) and simple to use. The execution times provided in the following table are in CPU time. Thus, when we give an execution time of 30 minutes, the required time is actually close to 15 minutes using two processing cores.

Table 2 presents results of the principal methods known to date. For each graph, it indicates the lowest number of colors found by each algorithm. For TabuCol [20] the reported results are from [21] which are better than those of 1987. The most recent algorithm, QA-col (Quantum Annealing for graph coloring [36]), provides the best results

	2013 <i>H₂col</i>	LS 1987/2008 TabuCol [20, 21]	1999 HEA [15]	2008 AmaCol [17]	Hybrid algorithm		
					2010 MACOL[26]	2011 EXTRACOL [38]	nov. 2012 QA-col [36]
DSJC500.1	12	13	-	12	12	-	-
DSJC500.5	47	50	48	48	48	-	47
DSJC500.9	126	127	-	126	126	-	126
DSJC1000.1	20	-	20	20	20	20	20
DSJC1000.5	82	89	83	84	83	83	82
DSJC1000.9	222	227	224	224	222	222	222
flat1000_50_0	50	50	-	50	50	50	-
flat1000_60_0	60	60	-	60	60	60	-
flat1000_76_0	81	88	83	84	82	82	81

Table 2: Best coloring found

but is based on a cluster of PC using 10 processing cores simultaneously. Note that HEA [15], AmaCol [17], MACOL [26] and EXTRACOL [38] are also population-based algorithms also using TabuCol and GPX crossover or an improvement of GPX (GPX with $n \geq 2$ parents for MACOL and EXTRACOL and the GPX process is replaced in AmaCol by a selection of k color classes among a very large pool of color classes). Only QA-col has another approach based on several parallel simulated annealing algorithms interacting together with sharing criteria.

In a standard Simulating Annealing algorithm (SA), the probability of accepting a candidate solution is managed through a temperature criteria. The value of the temperature decreases during the SA iterations. A Quantum Annealing (QA) is a memetic algorithm without crossover and in which the local search is a SA. The only interaction between the individuals of the population occurs through a specific sharing process. A standard sharing process involves penalizing solutions that are too ‘close’ within the search space (the simplest sharing is to forbid having two similar solutions in the population). However, comparing a solution with all solutions of the population can be computationally expensive, especially for a large population. This is why QA-col compares a solution with only two others of the population: this comparison provides the solution-population distance. The value of this measure is integrated into the temperature value of each SA. If the solution-population distance is greater, then the temperature will be higher, and there will be a higher probability that the solution will be accepted. [30, 31] have also developed a specific population spacing management that resembles this one. However, there are different ways to calculate the distance between two solutions. For a set of solutions, [7] define *frozen same pairs* (respectively *frozen different pairs*) as pairs of vertices that are in the same color class (respectively in the different color class) for all solutions. In QA-col, authors use these definitions with a set of two solutions in order to define a distance between these two solutions as the difference between the number of frozen same pairs and the number of frozen different pairs.

Table 3 presents the results obtained with H'_2col , the first version of H_2col (without elites). This simplest version finds very good solutions (+1 color compared to the best known results) for difficult graphs within the literature. Only one method, QA-col, occasionally finds a solution with less color. The column **Iter_{TC}** indicates the number of iterations of the TabuCol algorithm (this is the stop criteria of TabuCol). The column **Success** evaluates the robustness of this method, providing the success rate: success_runs/total_runs. A success run is that which finds a legal k -coloring. The

Instances	k	Iter_{TC}	Success	Iter	Gene	Time
DSJC500.1	12	8000	15/20	2.5×10^6	158	0.2 min
DSJC500.5	48	8000	9/20	5×10^6	356	0.5 min
DSJC500.9	126	25000	10/20	15×10^6	317	2 min
DSJC1000.1	20	7000	6/20	6×10^6	472	0.9 min
DSJC1000.5	83	40000	16/20	137×10^6	1723	36 min
DSJC1000.9	223	30000	4/20	56×10^6	939	12 min
	222	60000	1/20	516×10^6	4304	131 min
flat1000_50_0	50	130000	20/20	1.1×10^6	5	0.5 min
flat1000_60_0	60	130000	20/20	2.4×10^6	9	1 min
flat1000_76_0	82	40000	19/20	152×10^6	1905	38 min
	81	30000	1/20	380×10^6	6333	118 min

Table 3: Results of H'_2col , the first version of H_2col algorithm (without elites)

average number of generations or crossovers performed during one success run is given by **Gene** value. The total average number of iterations of TabuCol preformed during H'_2O is $\mathbf{Iter} = \mathbf{Iter}_{TC} \times \mathbf{Gene} \times 2$. The column **Time** indicates the average CPU time in minutes of success runs.

H'_2col does not find the solutions each time for these graphs, but when it does, it is generally very rapid. For example, for the graph coloring DSJC1000.1 with 20 colors, recent results reported in the literature are:

- MACOL in 108 minutes (CPU 3.4GHz) [26]
- EXTRACOL in 93 minutes (CPU 2.8GHz) [38]

Our algorithm, H'_2col , achieves solutions in less than one minute (CPU 3.1GHz).

The main drawback of H'_2col is that it converges sometimes too quickly. In such instances it cannot find a solution before the two individuals in a generation become identical. The second version, H_2Col , adds more diversity while performing an intensifying role.

Table 4 shows the results obtained with H_2Col . Of primary important is that H_2Col finds solutions with fewer colors than all the best-known methods. Only the Quantum Annealing algorithm, using ten CPU cores simultaneously, achieves this level of performance. In particular, DSJC500.5 is solved with only 47 colors and flat1000_76_0 with 81 colors. We noted 1* the number of success runs when the solution is occasionally found, but on average this occurs in less than one in 20 runs. This is the case for graphs DSJC500.5 and flat1000_76_0 with 47 colors and 81 respectively.

The computation time of H_2Col is generally close to that of H'_2Col but the former algorithm is more robust with quasi-100% of success. In particular, the two graphs DSJC500.5 and DSJC1000.1 with 48 and 20 colors respectively are resolved each time, and in less than one CPU minute on average. Using a multicore CPU, these instances are solved in less than 30 seconds on average, often in less than 10 seconds. As a comparison, the shortest time reported in the literature for DSJC1000.1 is 93 minutes

Instances	k	Iter _{TC}	Success	Iter	Gene	Time
DSJC500.1	12	4000	20/20	3.8×10^6	483	0.3 min
DSJC500.5	48	8000	20/20	7×10^6	494	0.9 min
	47	12000	1*	20×10^6	850	3 min
DSJC500.9	126	15000	13/20	29×10^6	970	4 min
DSJC1000.1	20	3000	20/20	4×10^6	346	0.7 min
DSJC1000.5	83	40000	20/20	96×10^6	1200	16 min
	82	40000	2/20	548×10^6	6854	96 min
DSJC1000.9	223	30000	19/20	126×10^6	2107	32 min
	222	50000	1/20	1.4×10^9	14208	354 min
flat1000_50_0	50	130000	20/20	1.1×10^6	5	0.5 min
flat1000_60_0	60	130000	20/20	2.2×10^6	9	1 min
flat1000_76_0	82	40000	20/20	84×10^6	1052	16 min
	81	40000	2/20	716×10^6	8961	116 min

Table 4: Results of the second version of H_2col algorithm (with elites) with the indication of CPU time

for EXTRACOL with a 2.8GHz processor (and 108 minutes for MACOL with a 3.4GHz processor).

5 Analysis of diversification

In this section we perform several tests in order to analyze the role of diversification in the H_2col algorithm. We increase or decrease the dose of diversification within the H_2col algorithm. There are only two operators that lead to diversification: the GPX crossover and the population update process. In a first set of tests, we slightly modify the dose of diversification in the GPX crossover and analyze the results. In a second set of tests, we focus on the population update process: in H_2col , the two produced children systematically replace both parents, even if they have worse fitness values than their parents. If we lighten this rule, the diversification decreases.

5.1 Dose of diversification in the GPX crossover

Some modifications are performed on the GPX crossover in order to increase (as for the first test) or decrease (as for the second test) the dose of diversification within this operator.

5.1.1 Test on GPX with increased chance: random draw of a color classes number

In order to increase the levels of chance within the GPX crossover, one randomizes the GPX. One recalls (cf. section 3.2.2) that at each step of the GPX, the selected parent transmits the largest color class to the child. In this test, we begin by randomly

transmitting x color classes chosen from the parents to the child; after those x steps, one starts again by alternately transmitting the largest color class from each parent. x is the random level. If $x = 0$, then the crossover is the same as the initial GPX. If x increases, then the chance and the diversity also increase. To evaluate this modification of the crossover, we count the cumulative iterations number of TabuCol that one H_2col run requires in order to find a legal k -coloring. For each x value, the algorithm runs ten times in order to produce more robust results. For the test, we consider the 48-coloring problem for graph DSJC500.5 of the DIMACS benchmark. The figure 5 shows in abscissa the random level x and in ordinate the average necessary iterations number required to find a legal 48-coloring.

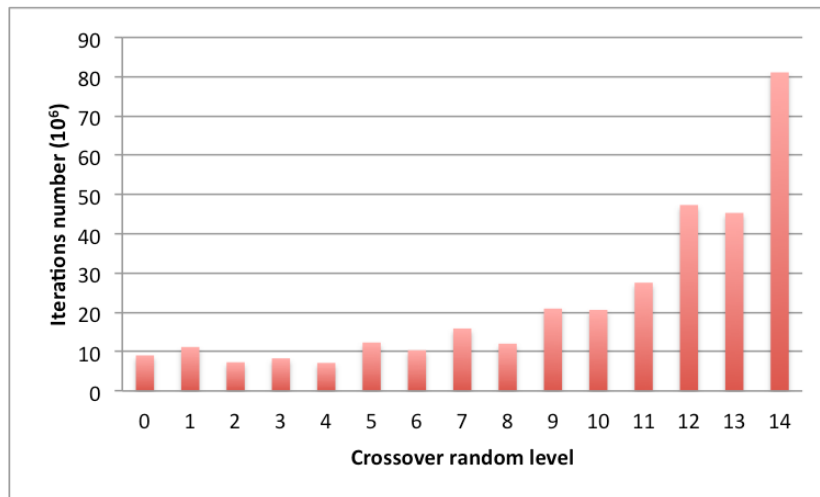


Figure 5: Average necessary iterations number to find a legal 48-coloring for DSJC500.5 graph in function of the randomness level; abscissa: x , the randomness level; ordinate: average iterations number

First, $0 \leq x \leq k$, with k the number of colors, but we stop the computation for $x \geq 15$, because with $x = 15$, the algorithm does not find a 48-coloring within acceptable computing time limit. This means that when we introduce too much diversification, the algorithm cannot find a legal solution. Indeed, for x high value, the crossover does not transmit the good features of the parents, so that the child appears to be a random initial solution. For $0 \leq x \leq 8$, the algorithm finds a legal coloring in more or less 10 million iterations. It is not easy to decide which x -value obtains the quickest result.

5.1.2 Test on GPX with decreased chance: imbalanced crossover and important different between the two parents

In the standard GPX, the role of each parent is balanced: they alternatively transmit their largest color class to the child. Of course, the parent, which first transmits its largest class, has more importance than the other; this is why it is an asymmetric

crossover. In this test, we give a higher importance to one of the parents. At each step, we randomly draw the parent that transmits its largest color class with a different probability for each parent. We introduce x , the probability of selecting the first parent; $1 - x$ is the probability of selecting the second parent. For example, if $x = 0.75$, parent 1 always has a 3 in 4 chance of being selected to transmit its largest color class (parent 2 only has a 1 in 4 chance). If $x = 0.5$, it means that both parents have an equal probability (a fifty-fifty chance) to be chosen; this almost corresponds to the standard GPX. If $x = 1$, it means that the child is a copy of parent 1; there are no more crossovers and therefore H_2col is a TabuCol with two initial solutions. When x get away from 0.5, the chance and the diversity bring by the crossover decrease. Figure 6 shows in abscissa the probability x and in ordinate the average number of necessary iterations required to find a legal 48-coloring (as in the previous test).

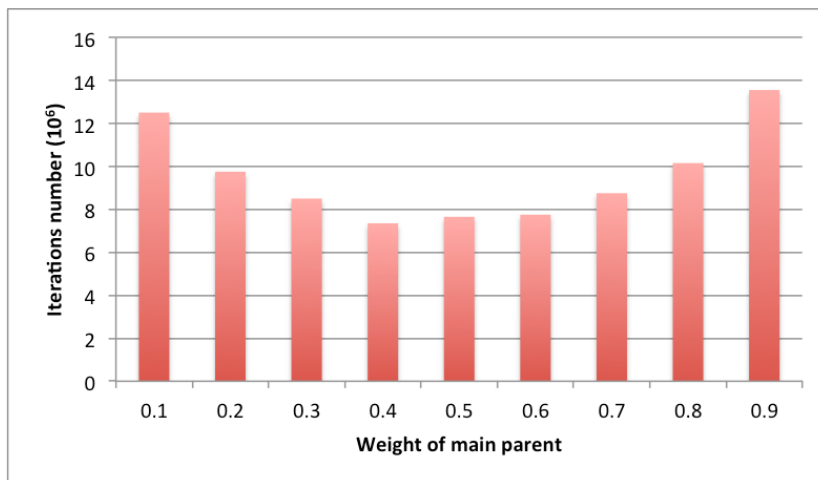


Figure 6: Average number of necessary iterations required to find a legal 48-coloring for DSJC500.5 graph in function of the imbalanced crossover; abscissa: x , probability to select the first parent; ordinate: average iterations number

First, it is evident that the results are symmetric according to x . The best results are obtained with approximately $x = 0.5$ ($0.4 \leq x \leq 0.6$). The impact of this parameter is weaker than that of the previous one: the control of the reduction of diversification is finer.

5.2 Test on parents' replacement: systematic or not

In H_2col , the two produced children systematically replace both parents, even if they have worse fitness values than their parents. We modify this replacement rule in this test. If the fitness value of the child is better than that of its parents, the child automatically replaces one of the parents. Otherwise, we introduce a probability x corresponding to the probability of the parents' replacement, even if the child is worse than his parents. If $x = 1$, the replacement is systematic as in standard H_2col and if $x = 0$, the

replacement is performed only if the children are better. When the x -value decreases, the diversity also decreases. Figure 7 shows in abscissa the parents' replacement probability x and in ordinate the average number of necessary iterations required to find a legal 48-coloring (as in the previous test).

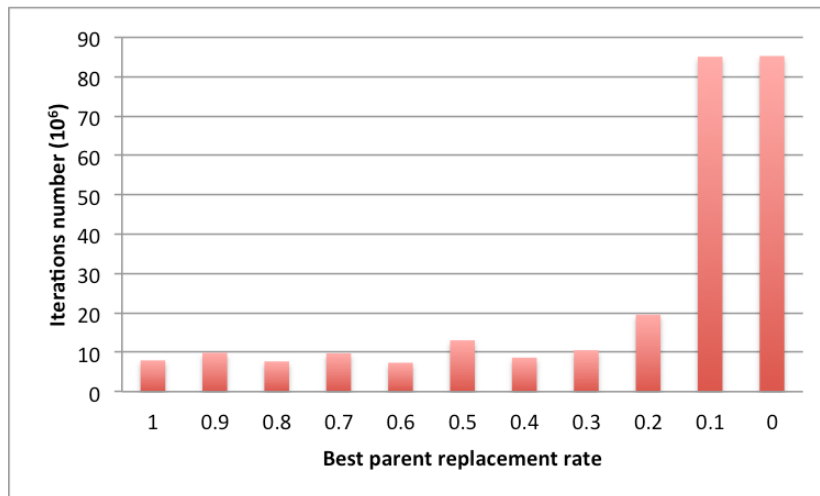


Figure 7: Average number of necessary iterations required to find a legal 48-coloring for DSJC500.5 graph in function of the parents' replacement policy; abscissa: parents' replacement probability; ordinate: average number of iterations

If the parents' replacement probability $x = 0$ or a very low 0.1, then more time is required to produce the results. The absence or the lack of diversification is shown to penalize the search. However, for a large range of values: $0.3 \leq x \leq 1$, it is not possible to define the best policy for x criteria. The dramatic change in behavior of H_2col happens very quickly around 0.2.

These studies enable us to better understand the role of the diversification operators (crossover and parent updating). Some interesting criteria are identified as the random level of the crossover or the imbalanced level of the crossover. We will integrate these criteria that we have studied in this paper into future algorithms in order to dynamically manage the diversity.

6 Conclusion

We have proposed a new algorithm for the graph coloring problem, called H_2col . It is a variation of a memetic algorithm with only two candidate solutions. This simplification has the great advantage of clarifying the role of the diversification and intensification operators and of more effectively managing the right 'dose' of diversification. H_2col combines a local search algorithm (TabuCol) as an intensification operator with a crossover operator (GPX) as a diversification operator, two main building blocks of Galinier and Hao's memetic algorithm [15]. The computational experiments, carried

out on a set of challenging DIMACS graphs, show that H_2col finds the best existing results, such as 47-colorings for DSJC500.5, 82-colorings for DSJC1000.5, 222-colorings for DSJC1000.9 and 81-colorings for flat1000.76_0, which have so far only been found by quantum annealing [36] with a massive multi-CPU.

We have performed an in-depth analysis on the crossover operator in order to better understand its role in the diversification process. Some interesting criteria have been identified, such as the crossover's levels of randomness and imbalance. Those criteria pave the way for our further research. We have generalized this optimization methodology, which is a specific memetic algorithm with only two individuals in its population. This Hybrid approach with 2 trajectories-based Optimization (H_2O) improves the local search algorithm through a crossover operator. The crossover inserts a dose of diversification that is then easy to manage.

References

- [1] Karen I. Aardal, Stan P.M. Hoesel, Arie M.C.A. Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4):261–317, 2003.
- [2] C. Avanthay, Alain Hertz, and Nicolas Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 2003.
- [3] Nicolas Barnier and Pascal Brisset. Graph Coloring for Air Traffic Flow Management. *Annals of Operations Research*, 130(1-4):163–178, 2004.
- [4] Claude Bernard. *Leçons de pathologie expérimentale*. J.-B. Baillière et fils, Paris, 1872.
- [5] D. Brélaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [6] Massimiliano Caramia and Paolo Dell’Olmo. Constraint Propagation in Graph Coloring. *Journal of Heuristics*, 8(1):83–107, 2002.
- [7] Joseph Culberson and Ian P. Gent. Frozen Development in Graph Coloring. *Theoretical Computer Science*, 265(1-2), August 2001.
- [8] Isabelle Devarenne, Hakim Mabed, and Alexandre Caminada. Intelligent neighborhood exploration in local search heuristics. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI ’06)*, pages 144–150. IEEE Computer Society, 2006.
- [9] Mohammad Dib. *Tabu-NG: hybridization of constraint programming and local search for solving CSP*. PhD thesis, University of Technology of Belfort-Montbéliard, December 2010.

- [10] Mohammad Dib, Alexandre Caminada, and Hakim Mabed. Frequency management in Radio military Networks. In *INFORMS Telecom 2010, 10th INFORMS Telecommunications Conference*, Montreal, Canada, May 2010.
- [11] N. Dubois and D. de Werra. Epcot: An efficient procedure for coloring optimally with Tabu Search. *Computers & Mathematics with Applications*, 25(10–11):35–45, 1993.
- [12] Nicolas Durand and Jean-Marc Alliot. Genetic crossover operator for partially separable functions. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 487–494, University of Wisconsin, Madison, Wisconsin, USA, 22–25 July 1998. Morgan Kaufmann.
- [13] C. Fleurent and J. Ferland. Genetic and Hybrid Algorithms for Graph Coloring. *Annals of Operations Research*, 63:437–464, 1996.
- [14] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Cosmin Porumbel. Recent Advances in Graph Vertex Coloring. In Ivan Zelinka, Václav Snásel, and Ajith Abraham, editors, *Handbook of Optimization*, volume 38 of *Intelligent Systems Reference Library*, pages 505–528. Springer, 2013.
- [15] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
- [16] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33:2547–2562, 2006.
- [17] Philippe Galinier, Alain Hertz, and Nicolas Zufferey. An adaptive memory algorithm for the k -coloring problem. *Discrete Applied Mathematics*, 156(2):267–279, 2008.
- [18] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of \mathcal{NP} -Completeness*. Freeman, San Francisco, CA, USA, 1979.
- [19] Jin-Kao Hao. Memetic Algorithms in Discrete Optimization. In Ferrante Neri, Carlos Cotta, and Pablo Moscato, editors, *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*, pages 73–94. Springer, 2012.
- [20] Alain Hertz and Dominique de Werra. Using Tabu Search Techniques for Graph Coloring. *Computing*, 39(4):345–351, 1987.
- [21] Alain Hertz, M. Plumettaz, and Nicolas Zufferey. Variable Space Search for Graph Coloring. *Discrete Applied Mathematics*, 156(13):2551 – 2560, 2008.
- [22] David S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Operations Research*, 39(3):378–406, 1991.

- [23] David S. Johnson and Michael Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA, 1996.
- [24] R.M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, USA, 1972.
- [25] F. T. Leighton. A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.
- [26] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.
- [27] A. Mehrotra and Michael A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal On Computing*, 8(4):344–354, 1996.
- [28] Jacques Monod. *Chance and necessity: an essay on the natural philosophy of modern biology*. Knopf, 1971.
- [29] M. Plumettaz, D. Schindl, and Nicolas Zufferey. Ant Local Search and its efficient adaptation to graph colouring. *Journal of Operational Research Society*, 61(5):819 – 826, 2010.
- [30] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms. In *9th European Conference on Evolutionary Computation in Combinatorial Optimisation (Evocop 2009)*, Tübingen, Germany, 2009.
- [31] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37:1822–1832, 2010.
- [32] Steven David Prestwich. Generalised graph colouring by a hybrid of local search and constraint programming. *Discrete Applied Mathematics*, 156(2):148–158, 2008.
- [33] El-Ghazali Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, September 2002.
- [34] Olawale Titiloye and Alan Crispin. Graph Coloring with a Distributed Hybrid Quantum Annealing Algorithm. In James O’Shea, Ngoc Nguyen, Keeley Crockett, Robert Howlett, and Lakhmi Jain, editors, *Agent and Multi-Agent Systems: Technologies and Applications*, volume 6682 of *Lecture Notes in Computer Science*, pages 553–562. Springer Berlin / Heidelberg, 2011.
- [35] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

- [36] Olawale Titiloye and Alan Crispin. Parameter Tuning Patterns for Random Graph Coloring with Quantum Annealing. *PLoS ONE*, 7(11):e50060, 11 2012.
- [37] D. C. Wood. A Technique for Coloring a Graph Applicable to Large-Scale Timetabling Problems. *Computer Journal*, 12:317–322, 1969.
- [38] Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290, 2012.
- [39] Nicolas Zufferey, P. Amstutz, and P. Giaccari. Graph Colouring Approaches for a Satellite Range Scheduling Problem. *Journal of Scheduling*, 11(4):263 – 277, 2008.